

# A TRIDENT SCHOLAR PROJECT REPORT

NO. 252

---

ADVANCED MACHINE CONTROLLER RETROFIT  
WITH ACOUSTIC EMISSION FEEDBACK

---



**DISTRIBUTION STATEMENT A**  
Approved for Public Release  
Distribution Unlimited

**UNITED STATES NAVAL ACADEMY  
ANNAPOLIS, MARYLAND**

This document has been approved for public  
release and sale; its distribution is unlimited.

20030604 137

USNA-1531-2

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

7 May 1997

3. REPORT TYPE AND DATE COVERED

4. TITLE AND SUBTITLE

Advanced machine controller retrofit with acoustic emission feedback

5. FUNDING NUMBERS

6. AUTHOR(S)

Roland R. Tink

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Naval Academy  
Annapolis, MD

8. PERFORMING ORGANIZATION REPORT NUMBER

USNA Trident Scholar project report  
no. 252 (1997)

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Accepted by the U.S. Trident Scholar Committee

12a. DISTRIBUTION/AVAILABILITY STATEMENT

This document has been approved for public release; its distribution is UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT: Current advanced milling uses Computer Numerical Control (CNC) to make complex shapes. The milling process uses a high speed rotating tool to remove material from a larger block until the desired workpiece shape remains. These shapes are frequently created by CAD and translated into tool path by CAM. This tool path defines the mill commands used to move the end mill to create the part. Since CNC technology today uses hardwired equipment, the mill command structure cannot be updated or modified without removing the controller. A possible solution is the implementation of the PC on the shop floor. PC's utilized as controllers for milling machines have been developed over the late 1980's and 1990's. A PC can easily handle the computing tasks of mill control, while also having the flexibility of being upgradable in implementing that control. New control codes or algorithms can be implemented by using new software, without the prohibitive cost of changing expensive control equipment. The additional advantage of utilizing the PC on the shop floor is the current research into Adaptive Controls (AC). AC represents an attempt by industrial researchers to optimize the milling process dynamically. Currently, mill technicians will optimize milling parameters such as depth of cut, feedrate, and spindle speed before any milling is done. During the milling process, these parameters remain unchanged. The disadvantage of this is the requirement to set each parameter at its most conservative setting so as to allow the mill to remove material safely under all end mill interactions. The mill operates at worst case conditions during the entire milling process. In this project, an Acoustic Emission (AE) sensor returned strain information to the PC. This strain information, which is referenced to material removal, and material removal rate, is indicated by a Root-Mean-Square (RMS) of the AE signal. From the RMS level, the PC adjusted feedrate accordingly. By controlling the feedrate, the PC can speed up or slow down the end mill trajectory. This process decreases overall milling time and/or improves surface quality by adjusting to the changing parameters during the milling process.

14. SUBJECT TERMS

adaptive control, milling, acoustic emission

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION  
OF REPORT

18. SECURITY CLASSIFICATION  
OF THIS PAGE

19. SECURITY CLASSIFICATION  
OF ABSTRACT

20. LIMITATION OF ABSTRACT

U.S.N.A --- Trident Scholar project report; no. 252 (1997)

ADVANCED MACHINE CONTROLLER RETROFIT  
WITH ACOUSTIC EMISSION FEEDBACK

by

Midshipman Roland R. Tink, Class of 1997  
United States Naval Academy  
Annapolis, Maryland



(signature)

Certification of Adviser Approval

Lieutenant Beth L. Pruitt, USN  
Department of Weapons and  
Systems Engineering




(signature)

7 May 1997

(date)

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade  
Chair, Trident Scholar Committee



(signature)

7 May 97

(date)

## Abstract

Current advanced milling uses Computer Numerical Control (CNC) to make complex shapes. The milling process uses a high speed rotating tool to remove material from a larger block until the desired workpiece shape remains. These shapes are frequently created by Computer Aided Design (CAD) and translated into a tool path by Computer Aided Machining (CAM). This tool path defines the mill commands used to move the end mill to create the part. Since CNC technology today uses hardwired equipment, the mill command structure cannot be updated or modified without removing the controller. A possible solution is the implementation of the Personal Computer (PC) on the shop floor. PC's utilized as controllers for milling machine have been developed over the late 1980's and 1990's. A PC can easily handle the computing tasks of mill control, while also having the flexibility of being upgradable in implementing that control. New control codes or algorithms can be implemented by using new software, without the prohibitive cost of changing expensive control equipment.

The additional advantage of utilizing the PC on the shop floor is the current research into Adaptive Controls (AC). AC represents an attempt by industrial researchers to optimize the milling process dynamically. Currently, mill technicians will optimize milling parameters such as depth of cut, feedrate, and spindle speed before any milling is done. During the milling process these parameters remain unchanged. The disadvantage of this situation is the requirement to set each parameter at its most conservative setting so as to allow the mill to remove material safely under all end mill interactions. The mill operates at worst case conditions during the entire milling process.

In this project, an Acoustic Emission (AE) sensor returned strain information to the PC. This strain information, which is referenced to material removal, and material removal rate, is indicated by a Root-Mean-Square (RMS) of the AE signal. From the RMS level, the PC adjusted feedrate accordingly. By controlling the feedrate, the PC can speed up or slow down the end mill trajectory. This process decreases overall milling time and/or improves surface quality by adjusting to the changing parameters during the milling process.

**Keywords:** Adaptive Control, Milling, Acoustic Emission

## Table of Contents

Problem Statement.....	Page 3
Background.....	Page 3
Design Concept.....	Page 5
Controller Retrofit.....	Page 6
Operation and Testing.....	Page 8
Sensor and Signal Conditioning.....	Page 9
Control Algorithm for Optimizing Feedrate.....	Page 13
Verification Test.....	Page 14
Conclusion.....	Page 15
Illustrations.....	Page 17
Bibliography.....	Page 24
Appendix A - Glossary.....	Page 25
Appendix B - Compensation Analysis.....	Page 26
Appendix C - P.A.C. S9204 AE Sensor Specifications	Page 29
Appendix D - LH0091 RMS-DC Specifications.....	Page 30
Appendix E - Communications Algorithm and Listing	Page 31
Appendix F - PIC16C71 Algorithm and Listing.....	Page 33
Appendix G - Feedrate Control Algorithm and Listing	Page 39
Appendix H - Mill Control Program.....	Page 42
Appendix I - Equipment List.....	Page 59

## Problem Statement

This project proposes to upgrade one mill by utilizing a personal computer to modify feedrate during the milling process. The feedrate determines how quickly a project can be completed by indicating how quickly the tool follows the tool path. The other factor in time to completion, depth of cut is predetermined by the part designer. Currently on the Computer Numerical Controlled (CNC) mill used in this project, the mill operator can operate a feedrate override that can speed up or slow down the tool feedrate, from 0-130% of the programmed feedrate. However, human reaction times are on the order of about a second, while a computer could make changes to the process in milliseconds. The reaction time to a mishap or for adjustment during the milling process can be reduced considerably. A reliable computer monitored and controlled milling center would allow for reduced manning during the tedious observation time of producing a part, as one machinist could tend several machines, and would allow a faster turnaround time in producing parts, with less breakage of material and tools.

## Background

Milling is a subtractive process by which material is removed from a larger stock of material leaving only the finished workpiece once the operation is complete. A visual description of how an end mill looks and interacts with the workpiece is located in figure (1).

Certain information is required to mill a workpiece, including knowledge of where the tool is in space, tool characteristics such as length and width, and knowledge of where the stock is in space. Hand operated mills, in operation for many decades, are not too difficult to use but require that a machinist assemble the data of tool location, size and shape, as well as workpiece location size and shape. The hand operator can then guide the mill in simple trajectories, e.g. straight lines or circles.

The need to create complex shapes led to computer numerical controller (CNC) machines. CNC mills use a software program to move the end mill along a toolpath, a predetermined sequence of locations over which material is to be removed. The system contains linear and sometimes circular interpolation routines to cut a path between points. This tool path is carried out by moving each axis the required amount to achieve coordinated motion along the path. Encoders provide position feedback which is used in closed loop motion control. An encoder quadrature count provides the controller with its "basic length-unit." This length is typically small, on the order of micrometers. The basic

unit length is the smallest increment of movement for an axis<sup>1</sup>.

The CNC mill uses a basic feedback loop with the input and output being the desired position and actual position respectively, the sensor being a position indicator, and the transfer function relating output to input describing the combination of the plant response, i.e., the amplifier and motor, and a compensator to alter that response. The compensator "tunes" the mill to respond in an acceptable fashion, i.e., no position error, and a quicker, smoother response. This is how CNC mills have operated on the shop floor for about two decades.

The CNC mill has automated the milling process in the sense that the tool path is programmed into the mill, and the operator observes milling as the CNC mill performs its work. After gaining experience with milling, a machinist can tell if a tool is experiencing too much or too little stress, and can increase or decrease the feedrate accordingly. Aggressive machinists push the mill to its operational limits, but this results in frequent tool and workpiece breakages. Machinists must be cautious in altering of feedrate, because they cannot react quickly enough to avert disaster.

Automation could solve that problem, as a computer can react far more quickly than a person. The trend toward automation has increased over the past few years toward unsupervised mills, capable of monitoring the milling process automatically. Currently, mills have only position feedback, and will only compensate for a sensed error in the tool path. There is no mechanism in place to allow the computer to supervise workpiece/tool status. While a machinist can see and hear the process, the computer can do neither. Sensors have been sought to evaluate in-process milling and provide the computer with accurate and relevant data about the workpiece/tool state. A sensor-based system ideally would allow the mill to monitor the material removal process, adjust machining parameters accordingly, and in emergencies, would stop the milling process without excessive damage. While a human can respond on the order of seconds, a computer controller can respond in milliseconds or even microseconds<sup>2</sup>.

This mill automation is labeled as Adaptive Control (AC)<sup>3</sup> and has been largely unsuccessful to this point. The problem facing engineers is what sensor or sensors can effectively monitor the milling process to both prevent catastrophic damage and optimize milling parameters, such as feedrate, depth of cut, and/or spindle speed. Various sensors have been put forth as possible solutions, force sensors, accelerometers, thermal sensors, acoustic emission (AE) sensors, etc. However, each of these sensors measures only one

---

<sup>1</sup>Koren, Y., "Design of Computer Control for Manufacturing Systems," *Journal of Engineering for Industry*, Vol. 101, August 1979, pp. 326-31.

<sup>2</sup>Sturges, R.H., "Monitoring Milling Processes Through AE and Tool/Part Geometry," *Journal of Engineering for Industry*, Vol. 114, February 1992, pp.8-14.

<sup>3</sup>Kannatey-Asibu, Elijah Jr. and Dornfeld D. A., "Quantitative Relationships for Acoustic Emission from Orthogonal Metal Cutting," *Journal of Engineering for Industry*, Vol. 103, August 1981, pp. 330-40.

parameter, either force, vibration, heat, or strain. No one parameter has proven to be the "magic" indicator of milling progress<sup>4</sup>. Sensors that monitor destructive events in milling, such as tool/workpiece breakage, are in use, but these systems do not prevent damage, but instead only minimize further damage once the event has occurred.

This project presents a system capable of reacting to data received from a sensor providing information on material removal rate by adjusting the spindle speed or feedrate. This control system can react to changing depths of cut by modifying the feedrate to prevent damage from occurring. Instead both the workpiece and tool were preserved as the milling parameters changed.

### Design Concept

This project began two years ago, under the sponsorship of the Naval Surface Warfare Center, (NSWC) Carderock Division. A mill at the Annapolis detachment was rendered defunct and turned over to experimentation and research. Initially the goal was to restore the mill to operation.

The mill required only a new mill controller to return to operation. The mechanics of the electric drive motors were still fully functional, and most peripheral equipment was available. What began as a project to restore the mill to operation became an enterprise aimed at enhancing the mill through experimental techniques.

Current mill controllers are hard-wired; that is to say they can only perform one function, controlling the tool during milling. To optimize or adjust the controller is difficult and costly, due to the fact that mill controllers are not reprogrammable, and instead must be rebuilt. While the mechanical parts of a mill can last for decades, the electronic controls become obsolete very quickly. The introduction of a mill controller that could be reprogrammed and upgraded as easily as a PC would result in a mill that remains useful far longer than current CNC mills.

The introduction of a new flexible controller, one that is PC based, allows for the introduction of new control methods. Now new sensors can be implemented with the mill, and control of the mill is passed to the computer. With the processing power of the computer, reaction times and algorithm complexity are greatly enhanced.

NSWC procured a computer controller, a Galil DMC-1060. This controller fits inside a regular PC and can act as a fully functional CNC controller. The mill was refit using the new controller, and a control algorithm was developed to utilize sensor data in the milling process.

---

<sup>4</sup>Ibid.



## The Controller Retrofit

In the last two decades, PC's have become more common in the workplace. Processing speed and dynamic memory have both increased exponentially since the 1980's. Because of this tremendous computing power available, the feasibility of using a PC on the shop floor has been recognized. Now, a controller that once took up a small cabinet can be placed inside a PC through an expansion slot. This project utilizes that technology. Using a Galil DMC-1060 computer control board, the PC can now be utilized as a computer controller for a fully functional milling machine.

The Galil DMC-1060 is an expansion card placed within a PC to enable it to control the mill. The DMC-1060 has the capability to control up to six axes at one time. This requires two expansion cards, with each card controlling three axes. The cards are referenced by the computer through port addressing. The address of the DMC-1060 is 1000 decimal, or 3E8 hex. Through this address, called the read/write address, and the control address, which is 1001 decimal, or 3E9 hex, the PC can interface with the controller. The PC can download commands to the controller and allow the microcomputer on the controller to buffer and execute the commands. This allows multi-tasking within the PC as it does not directly execute controller functions. A block diagram for the PC interface with the mill is located in figure (2).

Once the control board has been interfaced by the PC port, it processes the commands and executes them by use of an intermediary, the ICM-1100 board. The ICM-1100 is an external board that connects to the various devices on the mill. Encoders, inputs, outputs and control signals all connect to the ICM-1100. The ICM-1100 receives commands from the DMC-1060 by way of digital logic. Digital logic utilizes logic "yes" or logic "no" signals, by sending either 5 volts or 0 volts respectively. These signals are then sent to screw-type terminals on the ICM-1100 which connect to the various mill devices.

The mill is controlled entirely through digital inputs. This results in a safe, easily defined operation that requires little electrical power. The axis drive motors and spindle motor are controlled through a digital-to-analog signal based on a  $\pm 10$  volt range. This  $\pm 10$  volt digital to analog signal is generated by the Galil board to execute mill movement commands. The signal is transmitted to an amplifier which increases current to the motor proportional to the voltage of the signal. The amplifier runs the three-phase motor at 440 volts a-c. The sign of the control voltage indicates the direction of motor rotation. Thus, a +10 volt signal will result in the motor receiving maximum current from the amplifier to travel in the positive direction at maximum speed. The controller can compensate for the axis drive motor response by using a digital z-transform compensator within the computer. The digital z-transform compensation modifies the controller response by changing the mill transfer function, resulting in zero steady state position error and fast response with no overshoot.

The controller compensates for position error by receiving feedback from the encoders located on each axis. These encoders are phased signal encoders, which have two step signals ninety degrees out-of phase. These are called quadrature counts, and each

count is based on the input from both signals, and has four times the resolution of one signal, with the added benefit of the determination of direction coming from which signal goes high first. The encoders used in this project have 10,000 counts per rotation with direction indication. This means the encoder can detect a change of 0.036 of a degree of rotation. The counts are transferred to linear motion by a linear gain, from counts to inches which describes the gearing relationship of the axes. This gives the encoder extremely accurate position feedback. However, errors in converting linear position to encoder counts can arise from problems such as gear backlash. This error is nonlinear, and at best can be dampened out by an average backlash constant, determined experimentally. The average backlash was found to be approximately 20 counts, or 0.001". The mill controller compensates for this average backlash but variations from that average reduce the precision with which the mill operates by about 0.0005" on the linear axes.

While the encoder feedback and mill control represent the primary focus of the mill controller, there are peripherals on the mill which require control. These peripherals, such as the hydraulic pump, lube oil pump, coolant pump, etc., are all controlled digitally by the controller, connected to the ICM-1100 through computer cables. The controller sends a digital signal to the relays and powers up the pumps, clamps, and fans accordingly. This simple on/off control allows for easy troubleshooting of any malfunctions that can arise. These malfunctions are detected digitally as well. Sensors located on the peripherals indicate proper operation of that specific item. For example, once the hydraulic pump has been activated, a pressure sensor indicates whether or not minimum operational pressure has been obtained. Success is indicated digitally to the ICM-1100. When peripheral equipment is operating properly, milling operations are carried out. If a failure occurs, the ICM-1100 board will interrupt operation and halt milling if required. Thus the controller can monitor mill peripherals and detect any casualties that might arise.

Another position control aspect located on the mill is the use of limit and home switches. The mill has been fitted with two types of limit switches. The first type of limit switch sends a signal to the Galil board. This triggers the limit switch routine. The limit switch routine allows the user to move the axes in the direction away from the limit switch only. The motor axis can still function, but the programming requirements of the Galil board prevent commands to move the axis further in the direction of the limit switch. This occurrence is referred to as a "soft" limit; in other words, this limit is programmed and can be over-ridden. The second type of limit switch will shut down the mill when triggered. To recover from triggering this switch, the user must manually move the axis off of the limit switch. This "hard" limit cannot be overridden. The soft limit is located within the bounds of the hard limit so that the mill operator can simply move the axis away from the first limit switch as long as the mill has not yet triggered the hard limit. The limit switch provides safety from running the axis off of its track or colliding with the structure. The home switch is similar to a limit switch in that it is triggered in the same manner; however, the home switch indicates the "Zero" position for each axis. This location is important in that the mill has a set origin from which all positions are referenced. Since this origin is physical, it never changes, regardless of any computer malfunction or reset. The mill is homed on the origin and then carries out its program.

Mill control has now been reduced to one or two circuit boards located within a PC. These boards are much more economical than the hefty controllers still in use today, with a computer controller, personal computer and peripheral equipment costing thousands of dollars compared to the tens of thousands of dollars associated with new hardware controllers. The second advantage now available to manufacturers is the flexibility of the PC. New software can allow for newer control algorithms and more complex shapes, and also can be purchased or written at a fraction of the cost of procuring an entirely new controller. Older mills, such as the one used in this project, can now operate comparably with more modern mills because of the controller upgrade.

### **Operation and Calibration**

The Galil controller was installed into a Gateway 2000 Pentium 90 MHz computer, with a default address of 1000 decimal. Communication with the Galil board, consisting of writing to and reading from this Galil address, was accomplished through a C-language program.

A thorough test program established successful communication with the Galil controller. The connection of the mill required assignments of digital inputs and outputs for the mill functions. Again, a thorough test program established satisfactory operation of the peripheral features.

Mill drive motors were connected to the analog position outputs of the Galil controller. Frequency and step analysis of each axis was conducted to determine the transfer function of the mill. The transfer function of the mill indicates the response of the mill to a given command. This response is frequently very slow and can be sped up by the introduction of a compensator. The Galil board has a programmable compensator which accomplished the compensation. However, values for the compensator needed to be determined. The compensator is divided into three parts, a proportional, integral, and derivative compensator. The compensator that proved easiest to control was the proportional compensator. This compensator increased the error signal of the encoder position feedback from the current desired position. As the error in position was reduced, the control voltage decreased. The motor systems accounted for latent error due to the characteristics of the motor. A detailed account of the compensation analysis and results are documented in Appendix B. Further compensation could be done, but at a greater risk to possible instability in the motor. Since the ultimate aim of this project was the acoustic emission feedback control, conservative compensation was used.

Next, the conversion values for the axes were determined. The Galil board receives encoder counts as position feedback, but these counts must to be referenced to linear distances. A measuring device accurate to 0.0001" was utilized in measuring the linear

conversion values for the X, Y and Z axes and the rotational conversion value for the rotational axis, henceforth referred to as the W axis.

Once the conversion values were set, the home or "zero" position was referenced for each axis. Since the limit switch has a very low positional resolution, a two part method is used in the homing of the mill. First the mill axis triggers the home switch. Then the mill will slowly travel until the index pulse of the encoder is triggered. The index pulse is a single pulse that triggers once every rotation of the encoder. The result is a constant value for the home position of each axes. Currently, the index pulse for the W axis encoder is not functioning, and that limits the precision of the initial position of the W axis to degrees and not thousandths of a degree. However, changes in position can be measured to about 0.0001 degrees on the W axis. Using external methods, the initial position of the W axis can be set, and then normal milling can ensue with all four axes.

The actual command sequence is a linear interpolated vector move. The tool starts at position A and is commanded to move to position B. The vector speed is indicated in the same command or changed dynamically as the program is run. The Galil controller then correlates each axis to its required feedrate to accomplish the straight line vector motion. The command buffer stores 511 position commands, and can be reloaded by the PC continuously, which allows for large NC programs.

Once the calibration and operation techniques were worked out, milling operation and sensor testing began.

### **Sensor and Signal Conditioning**

The opportunity to upgrade the mill created the need to evaluate and decide on a sensor to utilize in the optimization process. There are many types of sensors available, including thermal sensors, force sensors, accelerometers, and acoustic emission (AE) sensors.

Two types of sensors, the force sensor and the AE sensor, are popular for use in tool monitoring. A force sensor is used to gauge cutting force applied from the spindle to the workpiece. This force is a factor of feedrate and depth of cut. However, commercially available cutting force sensors are not sensitive enough to be used for small variations in the workpiece's surface. With the small tolerances involved in milling, the force sensor does not respond well.

The other sensor, the Acoustic Emission (AE) sensor, detects elastic strain waves that travel through solid objects. These strain waves are created from a variety of what are termed AE "events", e.g dislocation, grain boundary sliding, workpiece fracture, plastic deformation, etc. In milling, the plastic deformation of the workpiece at the tool workpiece interface is the main source of AE. AE can be collected through various

methods, such as count rate, amplitude distribution, frequency spectrum, autocorrelation, and root mean square (RMS). These techniques are used to detect the intensity, frequency, and energy of the AE events during chip formation, plastic deformation, tool wear, etc. Frequency analysis has become more useful of late, because modern Fast Fourier Transform (FFT) techniques have made close to real time calculation of frequencies possible. However, frequency analysis cannot be made fast enough to allow for reliable control, only process observation and modeling. A more robust method is to monitor a range of frequencies for tool wear, chip formation, etc. The RMS monitoring of AE yields an approximation of the energy content of the signal. The RMS value of an a-c signal is the equivalent energy of a d-c signal over the same time period. Since the energy of AE comes from plastic deformation of material and the signal is composed of energy bursts and continuous energy sources related to material removal rate, a relationship between the volume of material removed and energy level was determined by Kannatey-Asibu<sup>5</sup>:

$$W' = \sigma \epsilon' V$$

where:

$W'$  = energy

$\sigma$  = stress, assumed constant

$\epsilon'$  = strain rate, assumed constant

$V$  = volume

The energy of the AE signal is dependent on strain rate, which is also related to the rate of material deformation, the applied stress, and the volume of material being removed. This relationship suggests a sensor that can be applied in monitoring volumetric removal. Parameters affecting volumetric removal are feedrate, depth of cut, spindle speed, and tool radius. This sensor can be used in the control of feedrate, and can be located away from the direct contact between tool and workpiece, but located on the workpiece itself for best results.

The AE sensor offers a sensitive response to changes in volumetric removal. Since it is excessive removal that can most likely damage a workpiece or tool, this parameter is important in the monitoring of the milling process. The AE sensor is attached to the workpiece and picks up strain waves traveling throughout the workpiece. These strain waves resonate throughout the structure and travel at very high frequencies. To respond well to these frequencies, the AE sensor needs to have a high resonant frequency. The sensor chosen for this project, the Physical Acoustics Corporation S9204, has a resonant frequency range of 75 to 200 KHz as seen in Appendix C. A diagram of the AE sensor and its connection to the workpiece can be seen in figure (3). The sensor is vise-mounted onto the stock material. One limitation of the sensor is the length of wire which limits the rotation of the table. A 360 degree rotation results in the wire wrapping the workpiece and possible damage to the sensor.

---

<sup>5</sup>Ibid.

The sensor feeds voltage levels corresponding to the energy levels of strain waves traveling through the workpiece. This voltage is amplified by an impedance matched preamplifier. This preamplifier magnifies the signal from the level of millivolts to volts. Once the signal has been amplified, the signal is fed into a RMS-to-DC chip, which converts the random waveforms into an average root mean square (RMS) value of the signal in volts. The chip is a National Semiconductor LH0091 RMS-DC chip. The specification sheet for this chip is listed in Appendix D.

The average value determined by the RMS chip is calculated over a sampling time determined by an external capacitor. The larger the capacitor, the longer the sampling time for each average. The result is a smoother signal and elimination of signal spikes. The capacitor value currently in use with the RMS-DC chip is a 1  $\mu$ F capacitor. This capacitor gives a sampling time of about 1.8ms. The result is a RMS signal that indicates the average energy level and still allows each tooth interaction with the workpiece to be visible at a spindle speed of 2400 rpm.

The experimental setup had the AE sensor located on the side of a 4"x3"x2.5" aluminum blank vise gripped into place. The end mill cut into the 4"x3" face along the 4" axis. Figure (4) displays the experimental setup for the end mill, workpiece, and AE sensor.

Testing was done at different feedrates and depths of cut. The variation was used to establish an experimental baseline for acceptable AE levels to use in the control mechanism. The following depths of cut were tested: 0.125", 0.25", and 0.5". The following feedrates were used with each depth of cut: 7.5 inches per minute (ipm), 15 ipm, 30 ipm, and 60 ipm. A plotter was used to capture typical waveforms for the testing done at 0.25" depth of cut, and these graphs are available in figure (5a) - (8a). For the 0.125" and 0.5" depth of cut, only approximated average values captured from the Oscilloscope were recorded. The purpose of these depths of cut were to establish rough baseline voltage levels to control the milling process. Optimization was set for the 0.25" depth of cut. When cutting at 0.25" depth of cut, the maximum established feedrate was set at 30 ipm. This feedrate provided the best trade-off between surface finish and time to mill. Slower feedrates allowed for a better surface finish while faster feedrates degraded it. Refer to figures (5b) - (8b) to observe the surface finish at various feedrates.

From the data gathered, the two conclusions of earlier studies could be confirmed. First, depth of cut had a much greater impact in RMS levels than feedrate, with a fairly linear increase in RMS values. A doubling of the depth of cut led to an approximate doubling of the RMS value. The doubling of the RMS value corresponds to the doubling of the material removal rate. Second, feedrate does not affect RMS levels in a very consistent manner or one that is easily predictable. In some cases, the RMS went down with an increase in feedrate or the values were different with a change in feedrate. This is shown in the recorded graphs in figures (5a) - (8a). This information was



consistent with Diei and Dornfeld's earlier work<sup>6</sup>. The equation for RMS is<sup>7</sup>:

$$\text{RMS} = C \sin(\alpha) \left[ \tau_k b_1 U \left( t_1 \left[ \frac{\cos(\alpha)}{\sin(\phi)} \cos(\phi - \alpha) \right] + \frac{1}{3} (l + 2l_1) \left[ \frac{\sin(\phi)}{\cos(\phi - \alpha)} \right] \right)^{1/2} \right]$$

where:

- RMS = Root Mean Square of AE signal
- C = Proportionality constant (experimentally defined)
- $\alpha$  = Rake angle (angle of tool face incidence) - fixed by tool geometry
- $\tau_k$  = shear strength of workpiece (material property)
- $b_1$  = width of cut ( geometry during cutting )
- U = cutting velocity of spindle
- $t_1$  = feedrate
- $\phi$  = shear angle (angle at which primary part deformation occurs)
- l = chip-tool contact length
- $l_1$  = length of sticking zone

As is observed in the above equation, there are quite a few parameters that participate in the RMS value obtained from the milling process. If all parameters are made constant with the exception of feedrate we get:

$$\text{RMS} = K_1 (K_2 t_1 + K_3)^{1/2}$$

where K represents the combination of various constant values. It is observed that the RMS will increase proportional to the square root of the feedrate. Thus large changes in feedrate will have small effect on RMS.

Surprisingly, when testing with various feedrates was done, the resulting RMS levels did not indicate a general increase with feedrate. While volumetric increases affected RMS according to predictions, feedrate increases actually decreased the voltage level in some instances. While further evidence would be required to prove such a hypothesis, it is possible that the depth of each cutting surface was such that the material being cut did not require greater force than shallower cutting depths. The increased feedrate, given the already high initial feedrates tested, would not result in greater strain on the workpiece as material began to collect on the cutting face. It is possible that further material was able to shear off at less of an angle than at shallower depths of cut. The feedrates tested might be at a threshold of cutting properties that affected strain on the material in ways not accounted for earlier.

---

<sup>6</sup>Diei, E.N. and Dornfeld, D. A., "Acoustic Emission From the Face Milling Process--the Effects of Process Variables," *Journal of Engineering for Industry*, Vol.109, May 1987 pp.92-99.

<sup>7</sup>Kannatey-Asibu, Elijah Jr. and Dornfeld D. A., "Quantitative Relationships for Acoustic Emission from Orthogonal Metal Cutting," *Journal of Engineering for Industry*, Vol. 103, August 1981, pp. 330-40.

For the test, and other milling processes, this strange feedrate affect on RMS will not be a problem in regards to feedrate control. Since the reduction of feedrate to zero results in a reduction of RMS to zero, slowing the feedrate will still reduce RMS. Further testing will show by how much and when feedrate has the predicted effect on RMS.

The handler for RMS feedback is the Analog-to-Digital (A/D) processor in the PIC16C71. The control algorithm was developed using the PIC16C71 and the PC in conjunction with each other.

### **The Control Algorithm for Optimizing Feedrate**

The PIC processor was programmed with two distinct modes of operation. The first mode is the two bit parallel input from the computer. The second mode of operation is a dedicated A/D converter that conducts simple signal analysis and reporting. The constraints that exist for the operation of the PIC are limited communication bits for receiving and conveying information.

The parallel input for the PIC gives it the ability to be flexible when setting RMS threshold values. The user, through the communications program listed in Appendix F, sets two eight bit values, the low and high threshold values respectively. The user simply inputs the requested voltage levels for the RMS signal, and the program converts these voltages into binary values. These binary values are then downloaded into the PIC two bits at a time using the communications algorithm of the communications program.

The second mode of operation, the A/D conversion, is the heart of the signal processing of the AE RMS signal. The limits of the A/D sensor is 0-5 volts which is converted into an eight bit binary number. The eight bit resolution of a 5 volt value allows for a single step of 0.02 volts. The conversion time is 80  $\mu$ sec, which means about 22 samples for every RMS sample from the RMS-to-DC chip.

The algorithm for the PIC processor is listed in Appendix G. Essentially, the PIC processor first checks to see if the value captured from the A/D sensor is higher than the downloaded high threshold from the user. This represents an AE RMS level that is too high to be acceptable, either for surface finish or safety. If the captured value is too large, then the PIC sends a high threshold bit to the monitoring computer program in the PC. This high threshold bit remains activated regardless of future RMS levels until a PC acknowledgment signal is received. This prevents the monitoring program, which is not interrupt driven, from missing the high signal, which is periodic as the tool teeth engage



and disengage the workpiece. This is important given the reason for a high signal, namely that the RMS value indicates an undesirable or unsafe operating range. Once the computer acknowledges the signal, if the RMS signal warrants, the high threshold bit turns off.

The low threshold value worked to optimize feedrate, just as the high threshold value works to optimize surface quality and ensure safety. When the RMS value falls too low, the low threshold bit comes on, signaling the computer to increase feedrate. However, because the RMS value rises and falls with tooth interaction, the PIC processor counts the number of low values captured. Once the number of low values reaches 150 consecutive counts, the low threshold bit comes on. The count of 150 corresponds to the time of at least two cutting tooth interactions between the tool and workpiece for the 1" end mill moving at 2400 rpm.

To prevent feedrate changes while the tool was not interacting with the workpiece at all, a nominal threshold value is permanently programmed into the PIC. This threshold value requires the RMS level to be at least 0.5 volts in strength. The AE RMS level will, when the tool is cutting, quickly jump to a steady state RMS value which is stronger than 0.5 volts for the tests conducted here. When the tool is not interacting with the workpiece, the AE RMS level captures about 0.02 volts in background noise. The result is a dead zone which prevents tool optimization where the tool cannot be optimized, during feedforward movement, and prevents unwarranted increases in feedrate between tool passes and when approaching the workpiece.

The high and low bits of the PIC processor are read by a monitoring program, listed in Appendix E. This program, once triggered by either a high or low threshold bit, responds by altering the feedrate command for the mill. This new feedrate command takes effect immediately within the controller, which begins to either slow or speed up the tool feedrate to the new desired feedrate. If a high threshold bit is read, the mill begins to slow the feedrate by about 3 ipm. The feedrate is continuously reduced until the mill is operating at a programmed minimum feedrate, usually a very conservative value, in this case 7.5 ipm. If a low threshold bit is read, then the feedrate is increased by 0.3 ipm. The feedrate will increase until it reaches the maximum safe feedrate, 30 ipm in this case.

### **Verification Test**

To test the effectiveness of the feedrate controller, an increasing depth of cut experiment was set up. The mill was set up for an optimized run at 0.25" depth of cut, with a feedrate of 30 ipm. The blank was then rotated 3.5 degrees, resulting in an increasing depth of cut from 0.25" to about 0.5" over the 4" long face. At 30 ipm, a 0.5" depth of cut would normally result in a rough finish. The feedrate controller optimizes feedrate for the depth of cut to maintain a feedrate which still provides good surface

finish.

Predicated on the previous AE test results, which implied a weaker relationship between feedrate and AE RMS value than between depth of cut and AE RMS value, the changing depth of cut would result in the changes to the resulting feedrate. As the depth of cut decreases, the controller will increase feedrate to compensate for the lower AE RMS values. The feedrate will be increased enough to bring the AE RMS value above the lower threshold, and into the normal operating band. As depth of cut increases, the controller will decrease feedrate to compensate for higher AE RMS values, in this case, the controller will continue to decrease feedrate until the AE RMS value is lower than the high threshold, and again, into the normal operating band.

When the test was conducted, the RMS level quickly ramped up to trigger the high threshold. The reduced feedrate then lowered the RMS level to below the low threshold. The feedrate began to increase again. This ringing continued until the feedrate steadied out. Once the deeper cut was encountered, the RMS value increased, resulting in a conservative feedrate through the deep cut. The resulting surface finish was better than expected, and far better than running the mill through the blank without optimization.

The feedrate increased and decreased until it settled out at a steady state value. This apparent ringing approximated an underdamped response. As the gain in a control loop is increased, the response of the loop frequently vibrates as it settles on the final value. The feedrate control exhibited some of the same characteristics. It is a distinct possibility that the feedrate changes were too large for the motor to respond in time, given the slow time constants of the motor, which are about 0.1 sec, and that there was considerable overshoot of the optimized feedrate.

## Conclusions

The feedrate test was a success. The computer, when acting upon AE data received from the workpiece, was able to modify the feedrate according to changes in the workpiece, in this case an increasing depth of cut. The ringing encountered by the controller occurred as a result of the computer response time being much faster than the motor, as evident in figure (9). As the motor began to react, the computer overcompensated for the error. This problem can easily be dealt with by slowing the computer response to match or slightly exceed the motor response.

Another conclusion is that feedrate does have an effect on RMS level as seen in the RMS graph in figure (9). The only factor contributing to the decreasing RMS at the latter portion of the graph is decreasing feedrate. The anomalies seen during the standard

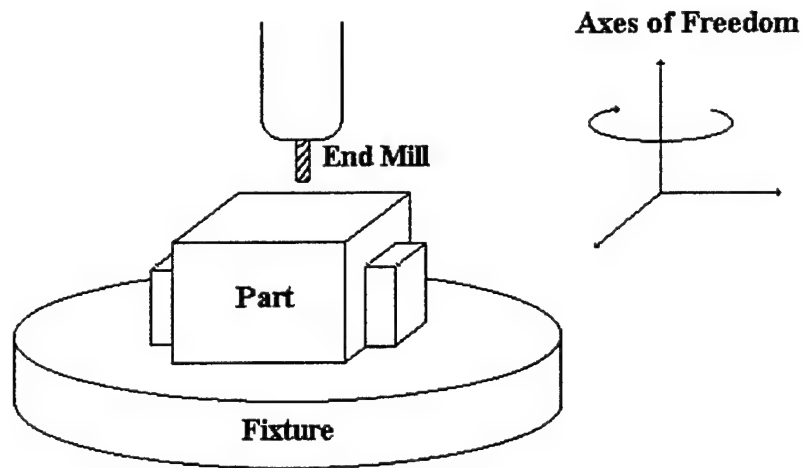
tests can be dampened out by the use of slower feedrates. For most milling applications, where steel is the milled medium, slower feedrates are the norm.

Anomalies on the part surface, such as waviness or ridges in the part, play a large part in the change in RMS voltage. As the depth of cut increased, there was a noticeable increase in RMS. The test put the controller through the worst case scenario, where the depth of cut changed radically. Now testing must be done to see if the controller can be fine tuned to small differences in material depth.

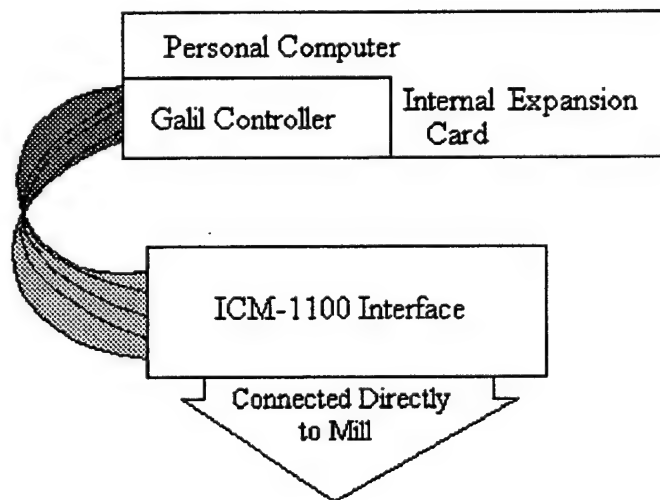
Changes in the workpiece are possible other than increasing depth of cut. Increasing and decreasing depth of cuts, surface waves, and different tool/workpiece geometries lead to new RMS values. In order to be adaptable, more complex control algorithm's as well as more testing is warranted for AE feedrate control.

At the very least, this new control system, utilizing Acoustic Emissions, can serve as a safety device for milling machines. The response of the controller can ensure that the tool and possibly the part can be preserved. However, this project is also a springboard to further development of autonomous mills, which could produce better parts with shorter turnaround time.

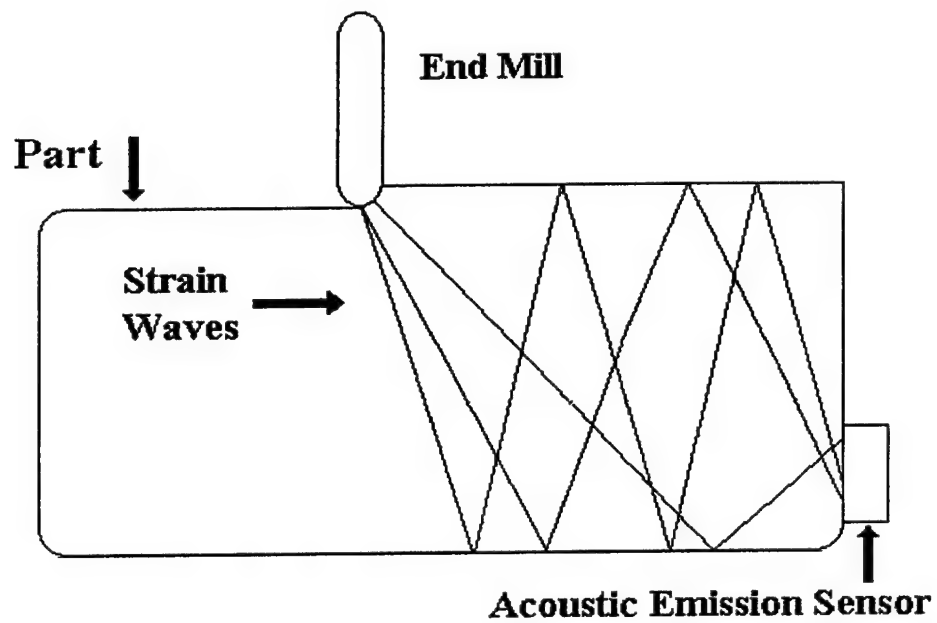
### Illustrations



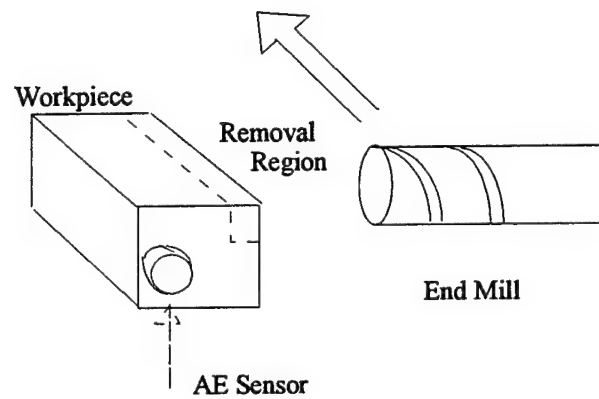
**Figure (1) End Mill with Workpiece**



**Figure (2) - Galil Controller Card Location and Connections**



**Figure (3) - Acoustic Emission Sensor Placement**

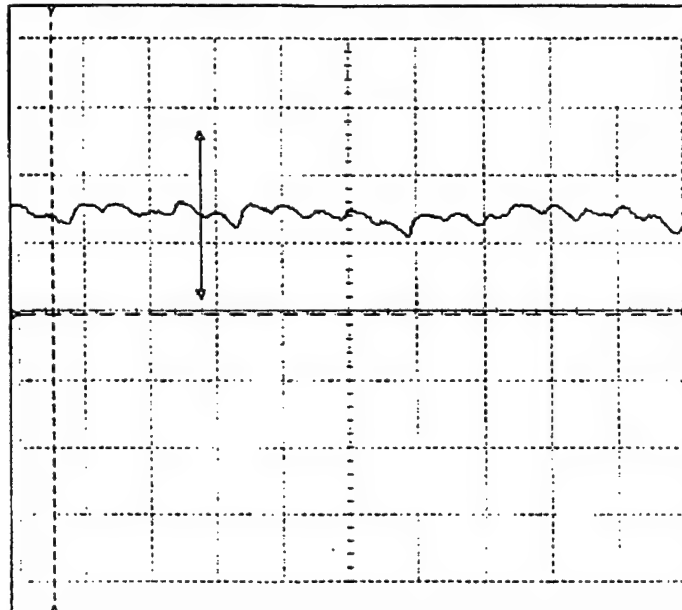


**Figure (4) Experimental Set-up**

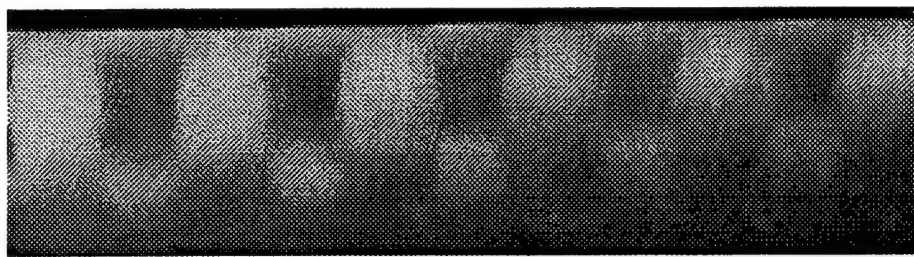
1.25 Volts/  
Div

CH1: 1.47DIV

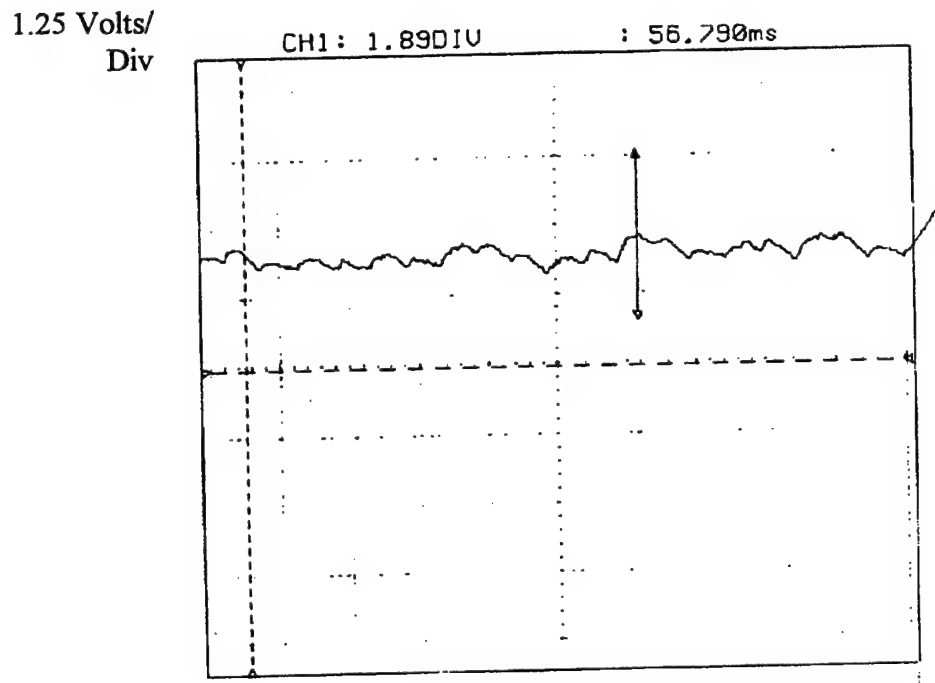
: 22.990ms



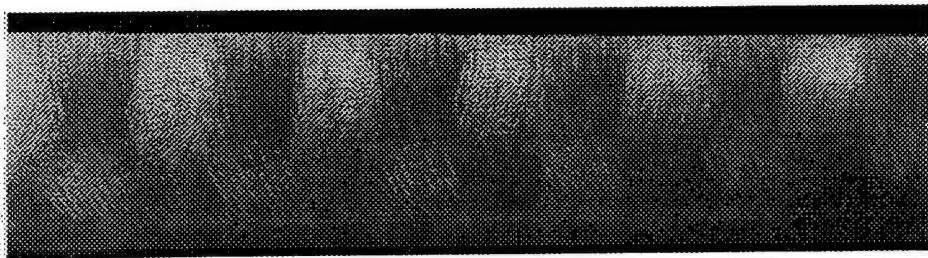
**Figure (5a) - RMS Values for 7.5 ipm feedrate**



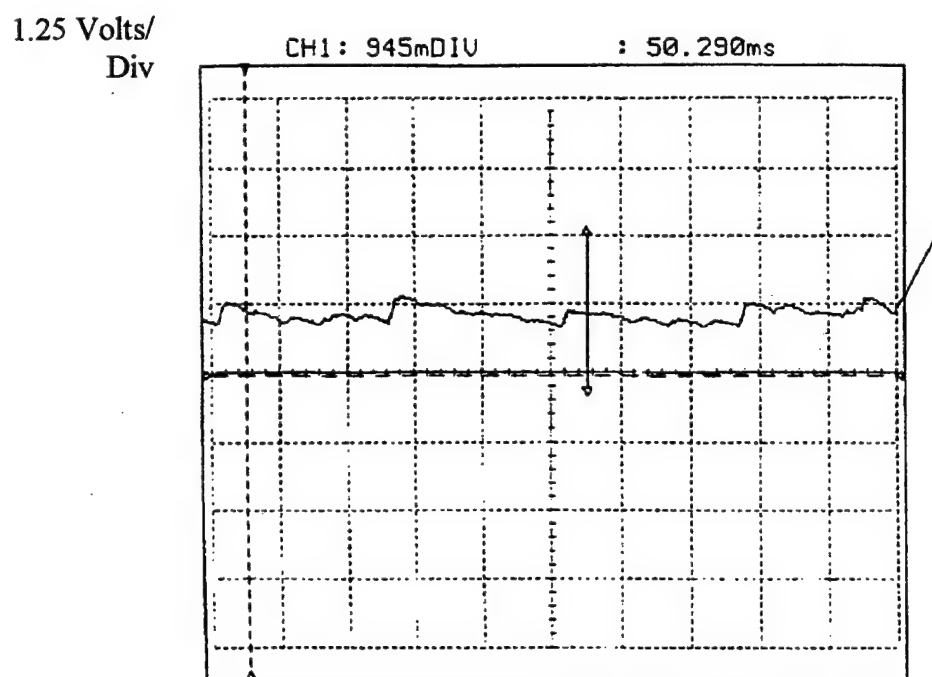
**Figure (5b) - Surface Finish of 7.5 ipm feedrate**



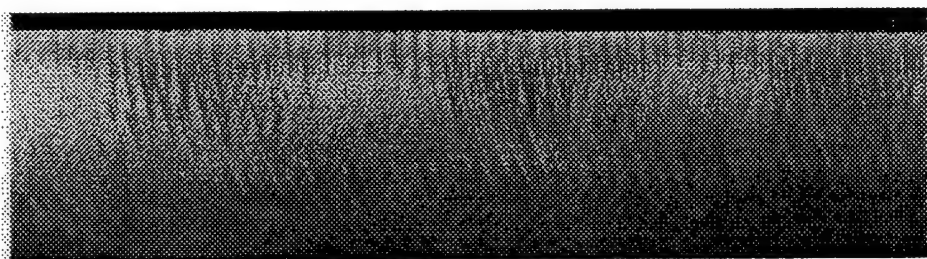
**Figure (6a) - RMS Values for 15 ipm feedrate**



**Figure (6b) - Surface Finish for 15 ipm feedrate**

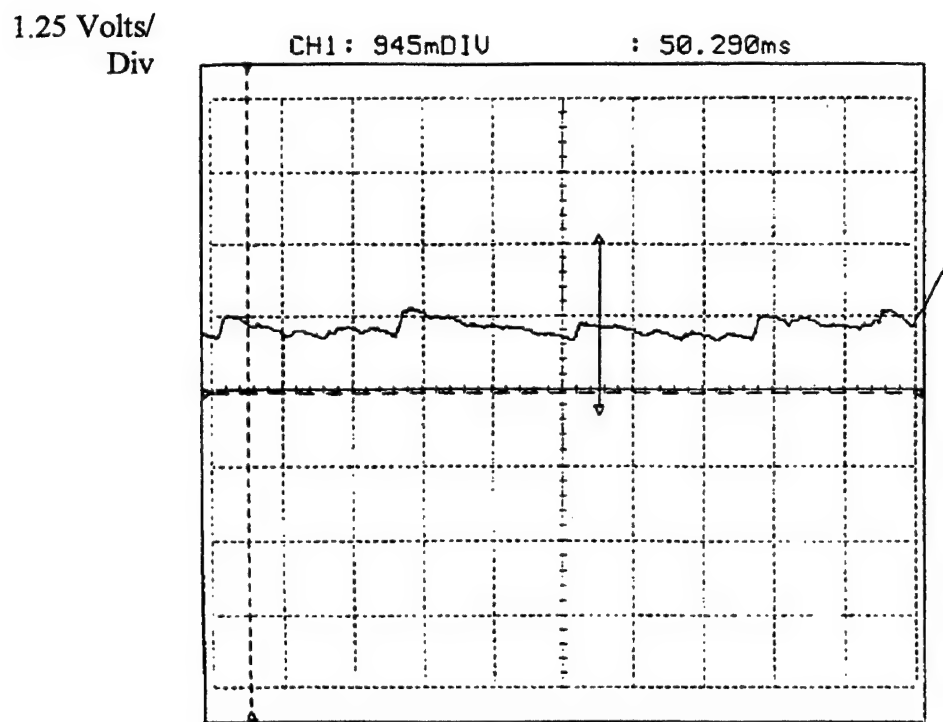


**Figure (7a) - RMS Values for 30 ipm feedrate**

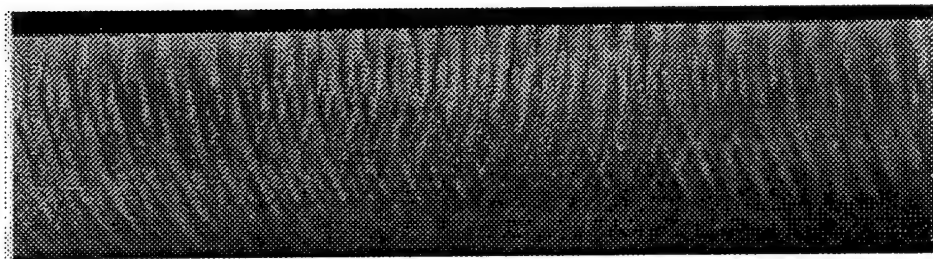


**Figure (7b) - Surface Finish for 30 ipm feedrate**

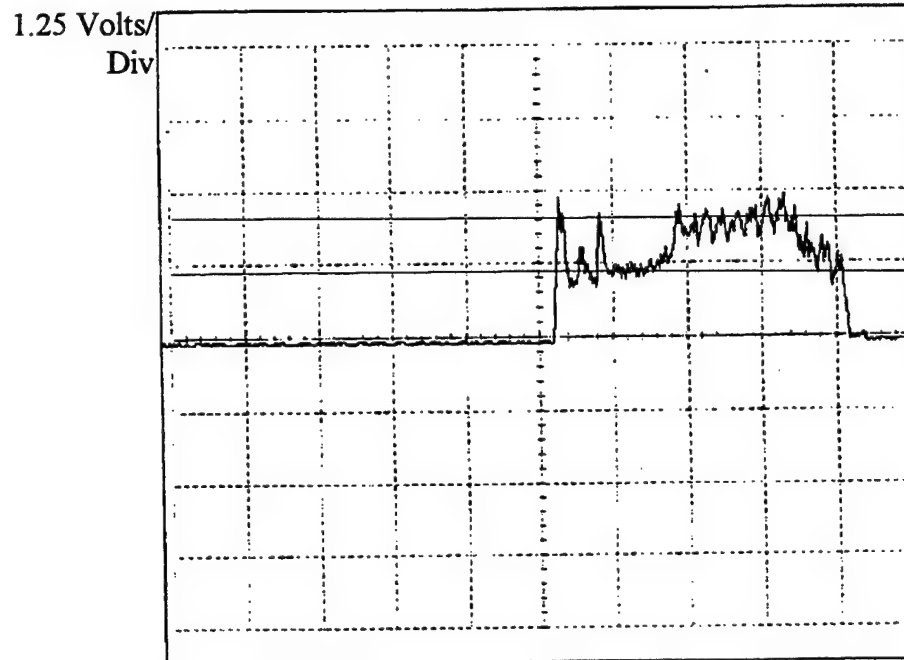




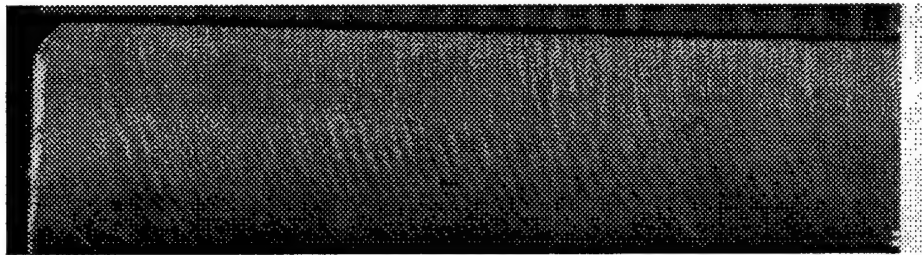
**Figure (8a) - RMS Values for 60 ipm feedrate**



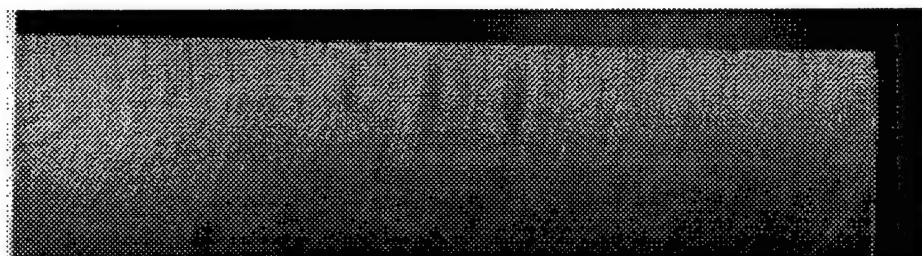
**Figure (8b) - Surface Finish for 60 ipm feedrate**



**Figure (9a) - RMS Values for Controller Test**



**Figure (9b) - Surface Finish for 30 ipm initial feedrate at 0.25" depth of cut**



**Figure (9c) - Surface Finish for 7.5 ipm final feedrate at 0.5" depth of cut**

### Bibliography

- Altintas, Y., "Prediction of Cutting Forces and Tool Breakage in Milling from Feed Drive Current Measurements," *Journal of Engineering for Industry*, Vol. 114, November 1992, pp.386-92.
- Ardekani, R. and Yellowley, I., "The Control of Multiple Constraints Within an Open Architecture Machine Tool Controller," *Journal of Manufacturing Science and Engineering*, Vol. 118 August 1996, pp.388-93.
- Chou, Jui-Jen and Yang, D. C. H., "On the Generation of Coordinated Motion of Five-Axis CNC/CMM Machines," *Journal of Engineering for Industry*, Vol. 114, February 1992, pp.15-22.
- Chryssolouris, G., Guillot, M., and Domroese M., "A Decision Making Approach to Machining Control," *Journal of Engineering for Industry*, Vol. 110, November 1988, pp.397-8.
- Diei, E.N. and Dornfeld, D. A., "Acoustic Emission From the Face Milling Process--the Effects of Process Variables," *Journal of Engineering for Industry*, Vol.109, May 1987 pp.92-99.
- Hutton, D. V., "A Noncontact Acoustic Emission Sensing System for Rotating-Cutter Machining Processes," *Experimental Techniques*, November/December 1991, pp.40-44.
- Kannatey-Asibu, Elijah Jr. and Dornfeld D. A., "Quantitative Relationships for Acoustic Emission from Orthogonal Metal Cutting," *Journal of Engineering for Industry*, Vol. 103, August 1981, pp. 330-40.
- Koren, Y., "Design of Computer Control for Manufacturing Systems," *Journal of Engineering for Industry*, Vol. 101, August 1979, pp. 326-31.
- Masory, O. and Koren, Y., "Stability Analysis of a Constant Force Adaptive Control System for Turning," *Journal of Engineering for Industry*, Vol. 107, November 1985, pp.295-300.
- Sturges, R.H., "Monitoring Milling Processes Through AE and Tool/Part Geometry," *Journal of Engineering for Industry*, Vol. 114, February 1992, pp.8-14.
- Thusty, J., "Dynamics of High-Speed Milling," *Journal of Engineering for Industry*, Vol. 108, May 1986, pp. 59-67.

## **Appendix A- Glossary of Terms**

**Acoustic Emission (AE)** - Strain waves formed from plastic deformation of material. Occurs in milling as a result of interaction between the end mill and the material being milled.

**Adaptive Control (AC)** - Method of controlling the mill machine by changing mill parameters such as feedrate, depth of cut, etc.

**Analog to Digital (A/D) Converter** - Converts an analog electrical signal into a digital binary value. The analog voltage is quantified into binary bits for storage or manipulation. The precision of the digital value depends on the number of bits available.

**Compute Numerical Controller (CNC)** - A mill that is programmed to execute a finite number of moves in a certain pattern. Allows for complex shapes in milling.

**Cutting Speed** - The speed of the tool in relation to the workpiece being cut. This speed is determined by the diameter of the tool in inches with the speed of the spindle in rpm.

**Encoders** - Devices that can sense position change and direction. These devices are used for position feedback from the mill.

**End Mill** - A type of cutting tool that primarily cuts from the end of the tool. This type of cutting tool is used for face milling for a smooth removal of material.

**Feedrate** - The speed at which material is fed to the end mill. The units are frequently in mm/rev or in/rev. This measures the length of material fed to the end mill for each revolution of the spindle.

**Root Mean Square (RMS)** - The average of a signal. The signal is first squared to get a positive value. The average of the squared term is then taken for a specified sampling time. The output is the square root of the averaged value.

**Spindle** - The rotating rod that connects the tool to the mill. The speed of the spindle in rpm's is used in the determination of cutting speed.

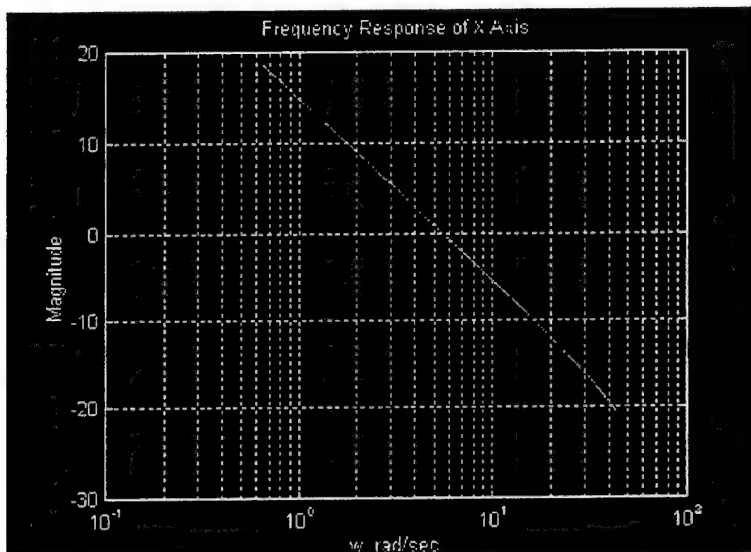
## Appendix B -

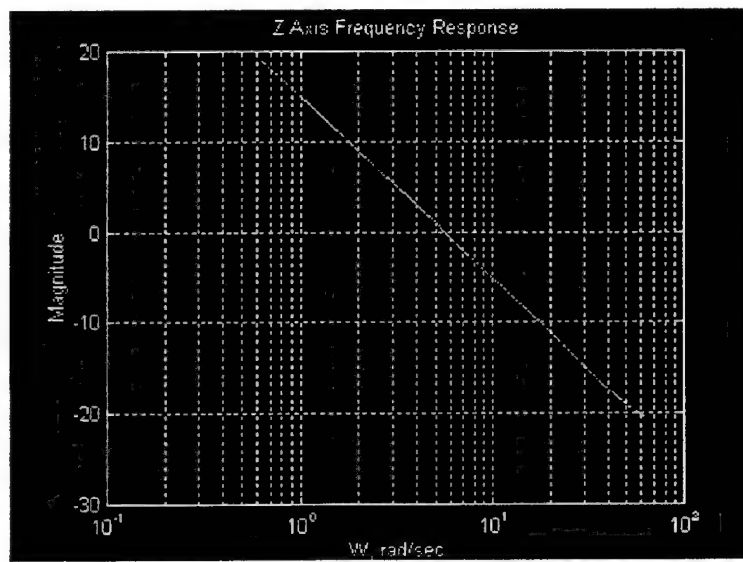
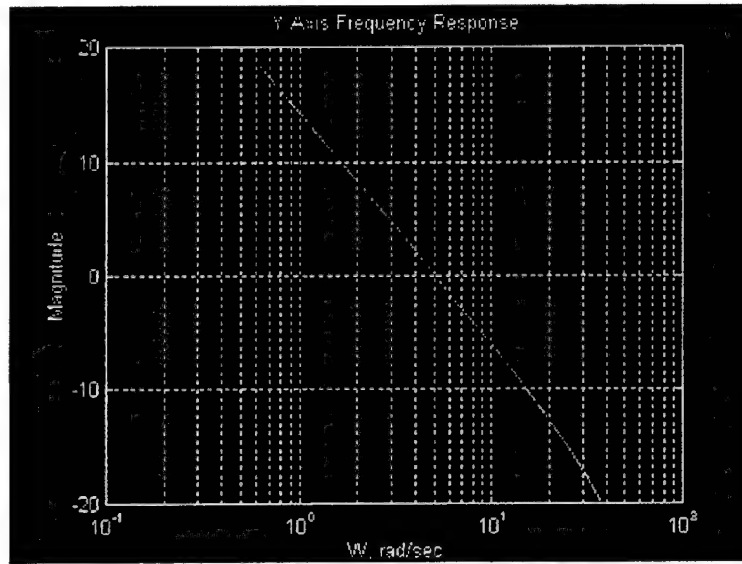
### Compensation Analysis

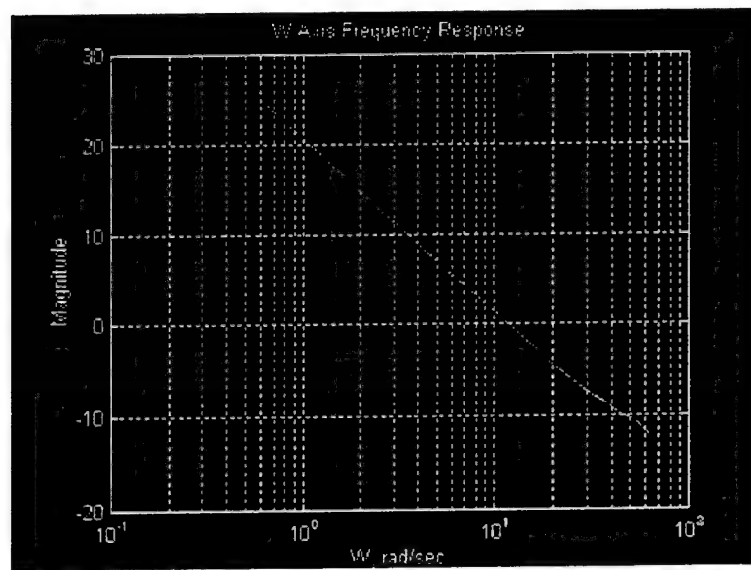
The compensation of the mill axis drives has two requirements. First, the tool must actually go to the commanded position of the motors. Second, the tool must not overshoot the ordered position. Within these two constraints, the mill is to be optimized to achieve commanded position as quickly as possible. To examine the drive motors, a frequency analysis test was performed on each axis. The plots of these analysis are listed below.

The plots reveal a transfer function pole at the origin of the drive motor. This is seen by the increasing magnitude of the response as the frequency is reduced to zero. This phenomenon indicates that each axis will have no error when it reaches its final commanded position.

The second aspect of the compensation technique is increasing the gain of the feedback in order to speed up mill drive response. In this case, the limiting factor in compensation is the requirement for no overshoot. Overshoot is the bypassing of the commanded position before settling out at the commanded position. A vibration that settles out is a clear example of overshoot. Since overshoot means the tool has removed too much material, this situation is not acceptable for milling. Testing with the axes gains resulted in the finding gains large enough to result in a fast response, but slow enough to prevent overshooting the commanded position. The gains used with the Galil controller were X: Gain of 2, Y: Gain of 2, Z: Gain of 2, and W: Gain of 1. The W axis had a much higher initial gain, as displayed in the frequency analysis graph, which resulted in a lower compensated gain.



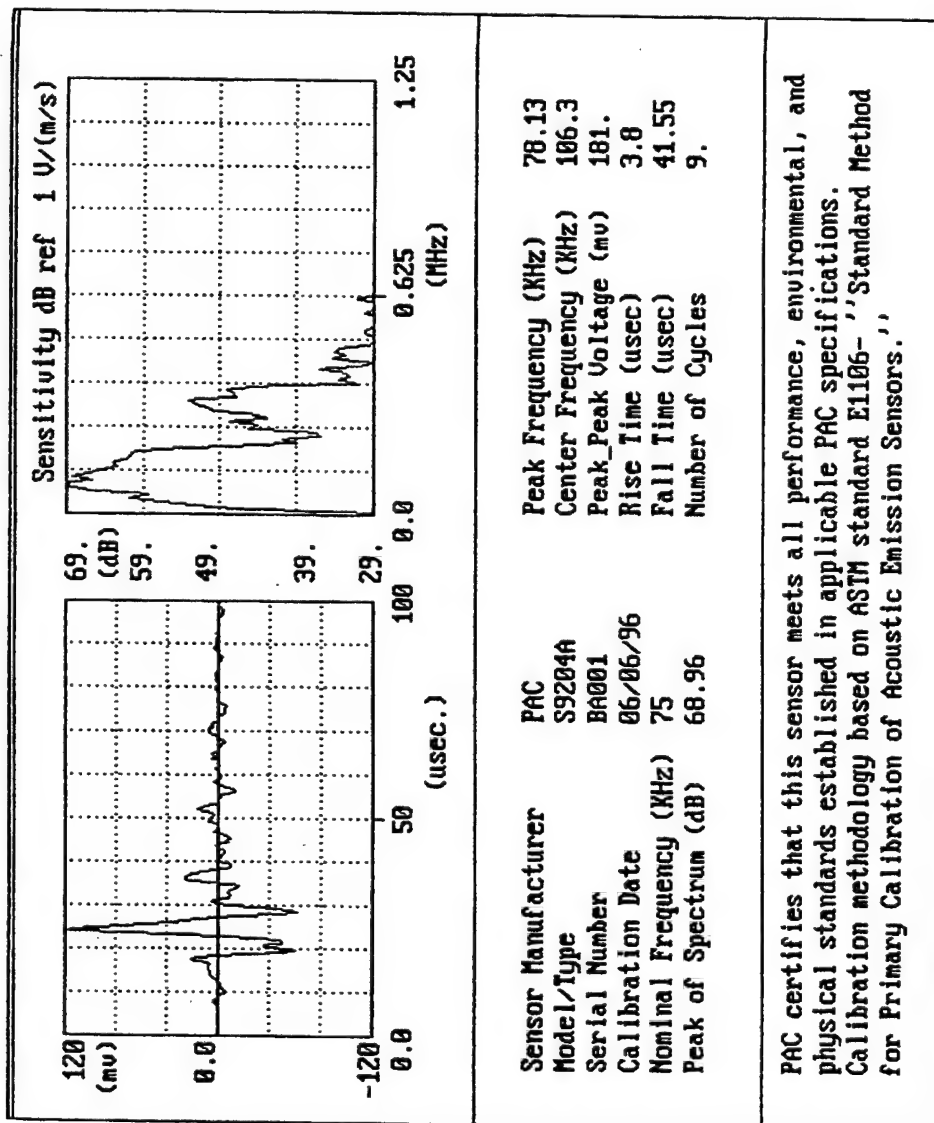




## Appendix - C

## S9204 AE Sensor Specifications

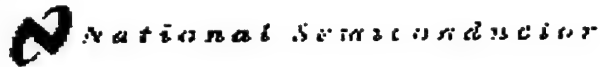
Specializing in Sound Technology ----- Physical Acoustics Corporation





## Appendix D -

## LH0091 RMS-DC Specifications



September 1993

## LH0091 True RMS to DC Converter

## General Description

The LH0091, rms to dc converter generates a dc output equal to the rms value of any input per the transfer function.

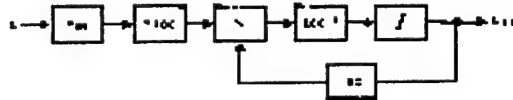
$$E_{OUT} = \sqrt{\frac{1}{T} \int_0^T E_{IN}^2 dt}$$

The device provides rms conversion to an accuracy of 0.1% of reading using the external trim procedure. It is possible to trim for maximum accuracy (0.5 mV ± 0.05% typ) for decade ranges i.e., 10 mV → 100 mV, 0.7V → 7V, etc.

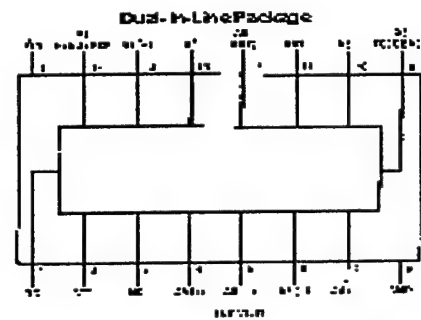
## Features

- Low cost
- True rms conversion
- 0.5% of reading accuracy untrimmed
- 0.08% of reading accuracy with external trim
- Minimum component count
- Input voltage to +15V peak for  $V_{CC} = +15V$
- Uncommitted amplifier for filtering, gain, or high crest factor configuration
- Military or commercial temperature range

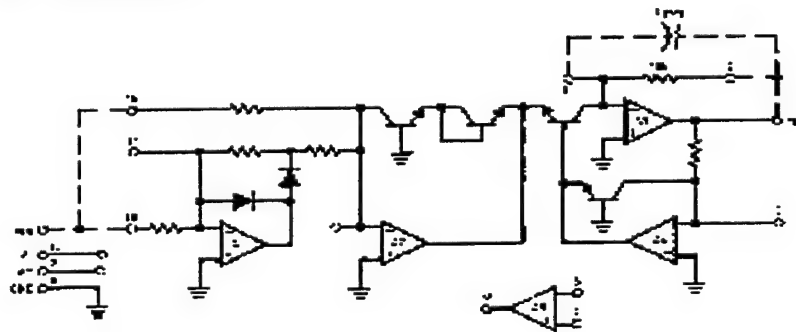
## Block and Connection Diagrams



Dual-In-Line Package  
Order Number LH0091CD  
See Package D 16D



## Simplified Schematic



LH0091 True RMS to DC Converter

## Appendix E -

### Communication Algorithm and Program Listing

The communications routine is set to download two bytes of information to the PIC16C71 processor. The downloaded bytes are the high and low thresholds of the AE RMS level. This enables different thresholds to be used in conjunction with the PIC processor, allowing for different materials to be milled.

The program asks the user for the low and high threshold values. These values are then converted to eight bit binary values. These values are then downloaded to the processor. The process for downloading the bytes to the processor is a standard parallel communication algorithm. First, the computer sets a ready-to-send (RTS) bit which is read by the PIC. The PIC, once the RTS bit is read, sets a ready-to-receive (RTR) bit. Once the computer has received the RTR bit, the RTS bit is reset, and the information is latched on a two bit port. The reset of the RTS bit tells the PIC that the information is latched on the port. The PIC resets the RTR bit, and reads the latched information.

#### Comport.CPP

```
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#include<math.h>

#define OPORT = 0x378
#define IPORT = 0x379
#define RTS   = 4
#define RTR   = 64

void send (int);

void main()
{
    float lovolt,hivolt;
    int lobits,hibits;

    outportb(0x378,0);
    printf("Enter low threshold voltage :\n");
    scanf("%f",&lovolt);
```

```

printf("\nEnter high threshold voltage :\ ");
scanf("%f",&hivolt);
if (lovolt >= hivolt)
    printf("\nLow value exceeded hi value\n");
lobits = (int)ceil(lovolt*255.0/5.0);
hibits = (int)ceil(hivolt*255.0/5.0);
printf("Low value is %x bits, Hi value is %x bits\n",lobits,hibits);
send(lobits);
send(hibits);
}

```

```

void send ( int v )
{
    int count=0;
    int outtmp=0;
    int in,out,rtr,rts;
    in = 0x379;
    out = 0x378;
    rtr = 64;
    rts = 4;
    delay(10);
    while (count < 4)
    {
        while (inportb(in) & rtr) {}
        outportb(out,rts);
        while (!(inportb(in) & rtr)) {}
        outtmp = (v>>(count*2)) & 3;
        printf("Sending %x Value\n",outtmp);
        outportb(out,outtmp);
        count++;
    }
}

```

## Appendix F -

### PIC16C71 Algorithm and Program Listing

The PIC16C71 processor is programmed to function in two modes. The first mode is as a receiver for the threshold information for the AE RMS level. The second mode is as an A/D converter with threshold comparison. The communications process is described in Appendix J.

Once the high and low operating thresholds have been set, the PIC processor continuously converts the incoming RMS signal at a rate of 80  $\mu$ sec per conversion. At each conversion the PIC goes through the following algorithm:

- 1) Test to see if the RMS level is higher than the threshold. If so, then set the high threshold bit going to the computer.
- 2) If the high threshold bit has not been acknowledged by the computer, then keep the high threshold bit set until the computer acknowledges it. This is done to guarantee that the computer will adjust the feedrate without possibly missing a high signal.
- 3) If the computer has acknowledged the high threshold bit, and the RMS level is lower than the high threshold, then reset the high threshold bit.
- 4) If the RMS level is lower than 0.5 volts do not test for a low threshold.
- 5) If the RMS level is higher than 0.5 volts, then test to see if the RMS level is lower than the low threshold value.
- 6) Count the number of times the RMS level is lower than the low threshold value. If the count exceeds 150, then set the low threshold bit. If the RMS level ever exceeds the low threshold value, reset the count. This is done to guarantee that at least two teeth will interact with the workpiece on a five tooth 1" end mill. The interaction must be below the threshold value to result in an increase in feedrate.
- 7) These steps are repeated until the chip is powered down.

#### Trident.ASM

```
fsr      = 4
indirect    = 0
status    = 3
intcon    = 11
trisa     = 85h
porta     = 5
adcon0    = 8
```

```

adcon1      = 88h
trisb       = 86h
portb       = 6
adres       = 9
wtemp       = 12
stemp       = 13
twobit      = 3
rts         = 4
rtr         = 3
low         = 14
hi          = 15
count       = 16
lowbit      = 2
hibit       = 3
intmp       = 17
counttmp    = 18
delay1      = 19
delay2      = 20
hireturn    = 7
locount     = 21
lowth       = 26

```

```

org 0
goto start
org 4
goto inth

```

```

inth
movwf wtemp
swapf status,w
movwf stemp
btfsc adcon0,1
goto conv
btfsc intcon,0
goto comms
done
swapf stemp,w
movwf status
swapf wtemp,f
swapf wtemp,w
retfie

```

```

conv

```

```
bcf adcon0,1
bsf adcon0,2
movf hi,w
subwf adres,w
btfsc status,0
bsf porta,hibit
btfss portb,hireturn
goto skip
btfss status,0
bcf porta,hibit
skip
movlw lowth
subwf adres,w
btfss status,0
goto nolong
movf low,w
subwf adres,w
btfss status,0
incf locount,f
btfsc status,0
clrf locount
movlw 150
subwf locount,w
btfss status,0
goto nolong
bsf porta,lowbit
decf locount,f
goto done
nolong
bcf porta,lowbit
goto done
```

```
comms
bcf intcon,0
bcf intcon,3
btfsc portb,rtss
goto setrec
movlw twobit
andwf portb,w
movwf intmp
movf count,w
movwf counttmp
```

```
movlw 4
subwf counttmp,f
btfss counttmp,7
goto secbyte
movf count,w
movwf counttmp
addwf counttmp,f
movlw 1
addwf counttmp,f
bcf status,0
loop1
rlf intmp,f
decfsz counttmp,f
goto loop1
rrf intmp,w
xorwf low,f
incf count,f
bcf portb,rtr
goto done
```

```
secbyte
movf counttmp,w
addwf counttmp,f
movlw 1
addwf counttmp,f
bcf status,0
loop2
rlf intmp,f
decfsz counttmp,f
goto loop2
rrf intmp,w
xorwf hi,f
incf count,f
bcf portb,rtr
goto done
```

```
setrec
bsf portb,rtr
goto done
```

```
start
movlw adcon1
movwf fsr
```

```
movlw 00000010b
movwf indirect
movlw trisa
movwf fsr
movlw 10011b
movwf indirect
movlw trisb
movwf fsr
movlw 11110111b
movwf indirect
movlw 0
movwf portb
movwf count
movwf low
movwf hi
movwf porta
movlw 10000000b
movwf intcon
movlw 10100001b
movwf adcon0
movlw 255
movwf delay1
movwf delay2
```

```
loop3
del1
del2
    decfsz delay2,f
    goto del2
movlw 255
movwf delay2
decfsz delay1,f
goto del1
bsf intcon,3
movlw 8
subwf count,w
btfss status,2
goto loop3
```

```
movlw trisb
movwf fsr
movlw 0
movwf indirect
```



```
movwf portb  
movwf porta  
movlw 11000000b  
movwf intcon  
movlw 10100101b  
movwf adcon0  
movlw trisb  
movwf fsr  
movlw 255  
movwf indirect
```

```
loop4  
goto loop4
```

## Appendix G -

### Feedrate Monitoring Program

The signals coming from the PIC processor control the feedrate via a monitoring program that continuously queues the PIC for high or low threshold information. If the program receives a high threshold, the following actions are taken:

- 1) The program sets the acknowledge bit to indicate to the PIC processor a successful receipt of the high threshold information.
- 2) The program decreases feedrate by 3 ipm, unless the feedrate is already at 7.5 ipm. Then the feedrate remains at 7.5 ipm.

If the program receives a low threshold bit from the PIC processor, then the program will increase feedrate by 0.3 ipm, unless the feedrate is already at 30 ipm. The maximum outputted feedrate will be 30 ipm.

#### Bittest.CPP

```
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
#include<string.h>

char *send(char *);

void main()
{
    int iport,oport;
    int hibit,lobit,retbit;
    unsigned char in;
    char *result,*cmd,*cptr;
    int vecspd;

    hibit = 32;
    lobit = 16;
    retbit = 8;
    iport = 0x379;
    oport = 0x378;
    Loop:
    outportb(oport,0);
    in = inportb(iport);
```

```

if (in & hibit)
{
    strcpy(cmd,"VS?;");
    result = send(cmd);
    cptr = cmd;
    while (*cptr)
    {
        if ((*cptr < 48) || (*cptr > 57))
            *cptr = 48;
        cptr++;
    }
    vecspd = atoi(result);
    if (vecspd > 2500)
        vecspd -= 1000;
    else
        vecspd = 2500;
    printf("Speed too high, vel is %i\n",vecspd);
    sprintf(cmd,"VS%i;",vecspd);
    send(cmd);
    outportb(oport,retbit);
}
if (in & lobit)
{
    strcpy(cmd,"VS?;");
    result = send(cmd);
    cptr = cmd;
    while (*cptr)
    {
        if ((*cptr < 48) || (*cptr > 57))
            *cptr = 48;
        cptr++;
    }
    vecspd = atoi(result);
    if (vecspd < 10000)
        vecspd += 100;
    else
        vecspd = 10000;
    printf("Speed too low, vel is %i\n",vecspd);
    sprintf(cmd,"VS%i;",vecspd);
    send(cmd);
}
goto Loop;
}

```

```
char *send(char *cmd)
{
    char *resp,*rptr;
    unsigned char chk;
    resp = (char*)malloc(sizeof(char)*20);
    rptr=resp;
    while (*cmd)
    {
        outp(1000,*cmd);
        cmd++;
    }
    delay(2);
    do
    {
        chk = inportb(1000);
        if (chk != 0xff)
        {
            *rptr = chk;
            rptr++;
        }
    } while (chk != 0xff);
    *rptr = NULL;
    return resp;
}
```

## Appendix H - Computer Control Algorithm and Program Listing

The computer control program for the Galil board, titled MILL.CPP, is used to address the Galil controller and provides an easy to use interface to operate the mill. The mill program follows a simple algorithm:

- 1) Send an initialization sequence to the Galil board, at address 1000. This sequence confirms communications and clears the memory buffer of the board.
- 2) Wait for user input. A prompt is displayed to the screen which allows the user to type in commands.
- 3) The command set for this program is:

**startup/shutdown** - Starts the peripheral devices on the mill, preparing the mill for operation. Shutdown reverses the process.

The peripheral are enabled in the following sequence:

- 1) Engage hydraulics
- 2) Engage the lube oil motor
- 3) Engage the fan motors
- 4) Disengage the Y-Axis brake
- 5) Engage the drive RGSO
- 6) Engage the drive enable switch

**hydraulics on/off** - Engages/disengages the hydraulics motor

**lube on/off** - Engages/disengages the lube oil motor

**fan on/off** - Engages/disengages the fan motor

**brake off/on** - disengages/engages the Y-axis brake

**coolant on/off** - engages/disengages the coolant motors

**hi/med/low speed** - sets the gearing for the spindle

**clutch** - clutches the gears from the spindle

**spindle start/stop** - enables the spindle drive motor

**terminal** - enters the user into the terminal portion of the program.

Here a user can directly command the mill to certain positions or feedrates.

**quit** - exits the program and shuts off the mill.

- 4) In addition, the mill program provides feedback as to the mill status, feedrates, positions. These feedbacks are handled by the computer and displayed to the user.

## Mill.CPP

```
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>

#pragma warn -def

struct postype {
    float x,y,z,w;
    postype *npos;
};

FILE *fptr;

//void interrupt IntInput(...);

//void interrupt (*oldfunc)(...);

int Initialize();

int EncodeCmd();

char *Convert ( int );

void tell(char*);

void InitIO();

char *send(char*);

void Terminal();

void start();

void shutdown();

void home(char);
```

```

void fgetpos(struct postype*);

void putpos(struct postype*);

void runpath(struct postype*, float*,float*,float*,float*);

void setpath();

void longrun(struct postype*);


void main()
{
// oldfunc=getvect(5);
// setvect(5,IntInput);
char *cmd,*result;
int commint=0;
struct postype parray;
float x1=0.0,y1=0.0,z1=0.0,w1=0.0;

clrscr();
if (Initialize())
{
printf("Board Ready\n");
InitIO();
while (commint != 1126)
{
commint = EncodeCmd();
switch (commint)
{
case 3114 : longrun(&parray);
break;
case 3022 : setpath();
break;

case 3807 : fgetpos(&parray);
putpos(&parray);
break;
case 3035 : runpath(&parray,&x1,&y1,&z1,&w1);
break;
case 3824 : Terminal();
break;
default : cmd = Convert(commint);

```



```

        result = send(cmd);
        if (strcmp(result, ".") != 0)
            tell(result);
        break;
    }

}
}
else
    printf("Error board not ready\n");
    printf("\nPress any key to continue...\n");
    while (!kbhit()) {};
// setvect(5,oldfunc);
}

int Initialize(void)
{
    int ready;

    inportb(1001);
    outportb(1001, 0x01 );
    outportb(1001, 0x80 );
    outportb(1001, 0x01 );
    outportb(1001, 0x80 );
    inportb(1001);
    outp(1001, 0x05 ); /* set up the FIFO for 128 bytes */
    outp(1001, 0xf0 ); /* bidirectionally */
    if ( (inportb(1001) & 16) == 0)
        ready = 1;
    else
        ready = 0;
    return ready;
}

int EncodeCmd ()
{
    int spec, count;
    count = 1;
    spec = 0;
    char *comptr;
    printf("\nEnter Command -->\n");
    gets(comptr);
    while (*comptr)

```

```

{
    spec = spec + ((int) *comptr)*count;
    count++;
    comptr++;
}
// printf("The int value is %i\n",spec);
switch (spec)
{
    case 4401 : printf("Setting Med Speed Gears\n");
                break;
    case 3543 : printf("Setting Hi Speed Gears\n");
                break;
    case 3114 : printf("Executing Long Run\n");
                break;
    case 3022 : printf("Setting Load File\n");
                break;
    case 3035 : printf("Executing Path Move\n");
                break;
    case 3807 : printf("Load Tool Path\n");
                break;
    case 9343 : printf("Starting Spindle\n");
                break;
    case 7965 : printf("Stoping Spindle\n");
                break;
    case 2249 : printf("Disengaging Gears\n");
                break;
    case 4477 : printf("Setting Low Speed Gears\n");
                break;
    case 3160 : printf("Beginning Startup Sequence\n");
                break;
    case 1126 : break;
    case 1709 : printf("Checking Inputs\n");
                break;
    case 9040 : printf("Engaging Hydraulics\n");
                break;
    case 10364 : printf("Disengaging Hydraulics\n");
                break;
    case 5398 : printf("Coolant On\n");
                break;
    case 6440 : printf("Coolant Off\n");
                break;
    case 1969 : printf("Fans On\n");
                break;
}

```

```

case 2635 : printf("Fans Off\n");
            break;
case 7779 : printf("Collet Clamped\n");
            break;
case 10755 : printf("Collet Unclamped\n");
            break;
case 7764 : printf("Pallet Clamped\n");
            break;
case 10740 : printf("Pallet Unclamped\n");
            break;
case 7652 : printf("Drives Enabled\n");
            break;
case 9044 : printf("Drives Disabled\n");
            break;
case 2636 : printf("Lube Oil On\n");
            break;
case 3396 : printf("Lube Oil Off\n");
            break;
case 1128 : printf("EMERGENCY STOP\n");
            break;
case 2705 : printf("RGSO Engaged\n");
            break;
case 3465 : printf("RGSO Disengaged\n");
            break;
case 637 : printf("Jog (Counts/Sec)\n");
            break;
case 4253 : printf("Y-Axis Brake Off\n");
            break;
case 3399 : printf("Y-Axis Brake On\n");
            break;
case 3824 : printf("Entering Terminal Mode\n");
            break;
case 4017 : printf("Shutting down mill\n");
            break;
case 1949 : printf("Homing Z-Axis\n");
            break;
default : printf("Unrecognized Command\n");
        }
return spec;
}

char *Convert ( int comm )
{

```

```
char *tmp;
char *xjg;
char *yjg;
char *zjg;
size_t allc;
switch (comm)
{
    case 4401 : strcpy(tmp,"OB11,1;OB13,1;OB14,0;");
                break;
    case 3543 : strcpy(tmp,"OB11,1;OB14,1;OB13,0;");
                break;
    case 9343 : strcpy(tmp,"OB12,0;");
                break;
    case 7965 : strcpy(tmp,"OB12,1;");
                break;
    case 2249 : strcpy(tmp,"OB11,1;OB13,1;OB14,1;");
                break;
    case 4477 : strcpy(tmp,"OB13,1;OB14,1;OB11,0;");
                break;
    case 1949 : home('z');
                strcpy(tmp,".");
                break;
    case 3160 : start();
                strcpy(tmp,"TI;");
                break;
    case 4017 : shutdown();
                strcpy(tmp,"TI;");
                break;
    case 1126 : strcpy(tmp,"OP16383;");
                break;
    case 1709 : strcpy(tmp,"TI;");
                break;
    case 9040 : strcpy(tmp,"OB1,0;");
                break;
    case 10364 : strcpy(tmp,"OB1,1;");
                break;
    case 2636 : strcpy(tmp,"OB2,0;");
                break;
    case 3396 : strcpy(tmp,"OB2,1;");
                break;
    case 5398 : strcpy(tmp,"OB3,0;");
                break;
    case 6440 : strcpy(tmp,"OB3,1;");
```

```

        break;
case 1969 : strcpy(tmp,"OB4,0;");
        break;
case 2635 : strcpy(tmp,"OB4,1;");
        break;
case 7779 : strcpy(tmp,"OB6,1;");
        break;
case 10755 : strcpy(tmp,"OB6,0;");
        break;
case 2705 : strcpy(tmp,"OB8,0;");
        break;
case 3465 : strcpy(tmp,"OB8,1;");
        break;
case 7652 : strcpy(tmp,"OB7,0;");
        break;
case 9044 : strcpy(tmp,"OB7,1;");
        break;
case 637 : printf("X Axis -->");
        gets(xjg);
        puts(xjg);
        printf("\nY Axis -->");
        gets(yjg);
        puts(yjg);
        printf("\nZ Axis -->");
        gets(zjg);
        puts(zjg);
        allc = 5*sizeof(char)+sizeof(xjg)+sizeof(yjg)+sizeof(zjg);
        tmp = (char*)malloc(allc);
        strcpy(tmp,"JG");
        strcat(tmp,xjg);
        strcat(tmp,"");
        strcat(tmp,yjg);
        strcat(tmp,"");
        strcat(tmp,zjg);
        strcat(tmp,"");
        break;
case 10740 : strcpy(tmp,"OB5,0;OB9,1;");
        break;
case 7764 : strcpy(tmp,"OB5,1;OB9,0;");
        break;
case 673 : strcpy(tmp,"UI1;");
        break;
case 4253 : strcpy(tmp,"OB10,0;");

```

```

        break;
    case 3399 : strcpy(tmp,"OB10,1;");
        break;
    default : strcpy(tmp,"");
}
return tmp;
}

```

```

void tell (char *resp)
{
    int    i=0,
           value[10],
           total,
           k=0,
           bit=0;

    if (strcmp(resp,"?") == 0)
        printf("\nCommand not accepted\n");
    /* else
    {
        while ((*resp != '.') && (i<10))
        {
            value[i] = (int)*resp - 48;
            if (value[i] < 0)
                value[i] = -1;
            i++;
            resp++;
        }
        total=i-1;
        for (i=total;i>0;i--)
        {
            if (value[i] != -1)
            {
                bit += value[i]*(int)pow(10,k);
                k++;
            }
        }
        bit = -bit-1;
        bit &= 0xff;
        if ( bit & 1 )
            printf("\nHydraulic Pressure\n");
        else
    }
    */

```

```

    printf("\nNo Hydraulic Pressure\n");
    if ( bit & 2 )
        printf("\nLube Oil Pressure\n");
    else
        printf("\nNo Lube Oil Pressure\n");
    if ( bit & 4 )
        printf("\nCollet Unclamped\n");
    else
        printf("\nCollet Clamped\n");
    if ( bit & 8 )
        printf("\nPallet Unclamped\n");
    if ( bit & 16 )
        printf("\nPallet Lowered\n");
    if ( bit & 32 )
        printf("\nPallet Ready\n");
    if ( bit & 64 )
        printf("\nPallet Clamp Pressure\n");
    if ( bit & 128 )
        printf("\nTable Clamp Pressure\n");
} */
}

void InitIO(void)
{
    char *cmd = "OP16383;";
    char *result;
    result = send(cmd);
    if (strcmp(result, ".") != 0)
    {
        printf("ERROR OUTPUT NOT SET\n");
        return;
    }
    // outp(1001,2);
    // outp(1001,4);
    // cmd = "EI32768,255;";
    strcpy(cmd, "MT-1,-1,-1,-1,-1;");
    send(cmd);
    strcpy(cmd, "KI0,0,0,0,0;");
    send(cmd);
    strcpy(cmd, "KP1,1,1,1,1;");
    send(cmd);
    strcpy(cmd, "KD0,0,0,0,0;");

```

```

send(cmd);
strcpy(cmd,"TL5,5,5,5,5,5;");
send(cmd);
strcpy(cmd,"AC999999,999999,999999,999999,999999;");
send(cmd);
strcpy(cmd,"DC999999,999999,999999,999999,999999;");
send(cmd);
strcpy(cmd,"SP5000,5000,5000,5000,5000;");
send(cmd);
strcpy(cmd,"TI;");
result = send(cmd);
tell(result);
}

```

```

/*
void interrupt IntInput (...)
{
    puts("Interrupted!");
}
*/

```

```

char *send(char *cmd)
{
    char *resp, *rptr;
    unsigned char chk;
    resp = (char*)malloc(sizeof(char)*20);
    rptr=resp;
    while (*cmd)
    {
        outp(1000,*cmd);
        cmd++;
    }
    delay(2);
    do
    {
        chk = inportb(1000);
        if (chk != 0xff)
        {
            *rptr = chk;
            rptr++;
        }
    } while (chk != 0xff);
    *rptr = NULL;
}

```



```

    return resp;
}

void Terminal()
{
    char *cmd,*result;
    printf("\nNow in Terminal Mode, enter commands direct to board.\n");
    printf("Use CAPS for commands. End command with ';' \n");
    do
    {
        printf("->\n");
        gets(cmd);
        if (strcmp(cmd,"QUIT") != 0)
        {
            result = send(cmd);
            if (strcmp(result,":") != 0)
                puts(result);
        }
    } while (strcmp(cmd,"QUIT") != 0);
    printf("\nExiting Terminal Mode\n");
}

void start ()
{
    char *cmd;
    strcpy(cmd,"OB1,0;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB2,0;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB4,0;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB10,0;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB8,0;");
    send(cmd);

```

```

    delay(1000);

    strcpy(cmd,"OB7,0;");
    send(cmd);
    delay(1000);
}

void shutdown ()
{
    char *cmd;
    strcpy(cmd,"OB7,1;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB8,1;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB10,1;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB4,1;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB2,1;");
    send(cmd);
    delay(1000);

    strcpy(cmd,"OB1,1;");
    send(cmd);
    delay(1000);
}

void home (char axis)
{
    char *cmd,*result;

    strcpy(cmd,"DL;#START;ZH=_HMZ;JP #START,ZH=0;STZ;AMZ;JG,,-500;");
    result=send(cmd);
    if (strcmp(result,":") != 0)
        puts(result);
}

```

```

strcpy(cmd,"FIZ;BGZ;EN;^Z");
result=send(cmd);
if (strcmp(result,":") != 0)
    puts(result);
printf("Now enter terminal mode\n");
}

```

```

void setpath()
{
    char *filename;
    printf("Enter filename ->\ ");
    gets(filename);
    printf("\n");
    fptr = fopen(filename,"r");
    if (fptr == NULL)
        printf("Error Opening File\n");
}

```

```

void longrun(struct postype *parray)
{
    struct postype *pptr,*lpptr;
    float x,y,z,w,x1,y1,z1,w1;
    int counter;

    x1 = 0.0;
    y1=7.5;
    z1=0.0;
    w1=0.0;
    pptr = parray;
    fscanf(fptr,"%c,%c,%c,%c\r",&x,&y,&z,&w);
    counter=0;
    while (!feof(fptr))
    {
        printf("Loading\n");
        while (counter < 50)
        {
            fscanf(fptr,"%f,%f,%f,%f\r",&x,&y,&z,&w);
            pptr->x = x;
            pptr->y = y;
            pptr->z = z;
            pptr->w = fabs(w);
            pptr->npos = (struct postype*)malloc(sizeof(struct postype));
            lpptr = pptr;

```

```

    pptr = pptr->npos;
    counter++;
}
lpptr->npos = NULL;
runpath(parray,&x1,&y1,&z1,&w1);
if (counter >= 50)
    printf("Paused...\n");
else
    printf("Done\n");
while (!getch()) {};
pptr = parray;
counter = 0;
}
}

```

```

void fgetpos(struct postype *parray)

```

```

{
    char *filename;
    struct postype *pptr,*lpptr;
    float x,y,z,w;
    int counter;

    pptr = parray;
    printf("Enter filename ->\ ");
    gets(filename);
    printf("\n");
    fptr = fopen(filename,"r");
    if (fptr == NULL)
        printf("Error opening file\n");
    else
    {
        fscanf(fptr,"%c,%c,%c,%c\r");
        counter=0;
        while (!feof(fptr) && counter < 50)
        {
            fscanf(fptr,"%f,%f,%f,%f\r",&x,&y,&z,&w);
            pptr->x = x;
            pptr->y = y;
            pptr->z = z;
            pptr->w = fabs(w);
            pptr->npos = (struct postype*)malloc(sizeof(struct postype));

```

```

    lpptr = pptr;
    pptr = pptr->npos;
    counter++;
}
lpptr->npos = NULL;
}
}

```

```

void putpos (struct postype *parray)

```

```

{
    struct postype *pptr;

    printf("X,Y,Z,W\n");
    pptr=parray;
    while (pptr != NULL)
    {
        printf("%.5f,%.5f,%.5f,%.5f\n",pptr->x,pptr->y,pptr->z,pptr->w);
        pptr = pptr->npos;
    }
}

```

```

void runpath (struct postype *parray, float *x1,float*y1,float*z1,float*w1)

```

```

{
    struct postype *pptr;
    float dx,dy,dz,dw;
    char *cmd;
    int counter,vel;
    char key=' ';

    pptr=parray;
    strcpy(cmd,"LM XYZW;");
    send(cmd);
    strcpy(cmd,"VS 10000;");
    send(cmd);
    vel = 10000;
    strcpy(cmd,"VA 100000;");
    send(cmd);
    strcpy(cmd,"VD 100000;");
    send(cmd);

```

```

    counter=0;

```

```

    while (pptr !=NULL)

```

```
{
    dx = pptr->x-*x1;
    dy = pptr->y-*y1;
    dz = pptr->z-*z1;
    dw = pptr->w-*w1;
    if (dw < 0)
        dw += 360.0;
    if (dw > 180.0)
        dw = -360 + dw;
    *x1 = pptr->x;
    *y1 = pptr->y;
    *z1 = pptr->z;
    *w1 = pptr->w;
    dx *= 19980;
    dy *= 20000;
    dz *= 20000;
    dw *= 13316.9;
    sprintf(cmd,"LI%.0f,%.0f,%.0f,%.0f",dx,dy,dz,dw);
    printf("Moving %.0f,%.0f,%.0f,%.0f\n",dx,dy,dz,dw);
    send(cmd);
    strcpy(cmd,"BGS;");
    send(cmd);
    strcpy(cmd,"AV;");
    send(cmd);
    pptr = pptr->npos;
}
}
```

**Appendix - I**  
**Equipment Parts List**

Kearney & Trecker Milwaukee-Matic NC mill

Micro Swinc NC controller

Gateway 2000 PC, Pentium 90MHz processor

Galil DMC-1060 controller

ICM-1100 interface board

S9204A AE sensor

National Semiconductor LH0091 RMS-DC chip

PIC16C71 8-bit CMOS Microcontroller